## IN THE SPECIFICATION

Please replace the third full paragraph on page 1 which begins with "In conventional runtime environments like" with the following amended paragraph:

In conventional runtime environments like the JAVA ~~Java™~~ R~~r~~untime E~~e~~nvironment (~~Java~~ JAVA is a trademark of Sun Microsystems of Palo Alto, California, United States) which enables the use and operation of computer programs coded according to the Java programming language utilizing a Java runtime environment, classes can be introduced when they are referenced by name in a class that already is executing within the runtime environment. While the entry point class of an application can require some individual processing exclusive of the class loading mechanism, subsequent attempts at loading other classes are performed exclusively by the class loader.

Please replace the third full paragraph on page 5 which begins with "Importantly, the custom class loader" with the following amended paragraph:

Importantly, the custom class loader can include a flag indicating whether the class has been replaced. In one aspect of the invention, the flag can be a dirty bit. Still, the invention is not limited in regard to the manner in which the custom class loader can determine whether an associated class has been replaced. In another aspect of the invention, the custom class loader can conform to the specification of a Java[™] version 1.2 delegation style custom class loader.

Please replace the first paragraph on page 10 which begins with "Figure 1 is an object" with the following amended paragraph:

Figure 1 is an object illustration of a system of cyclically dependent classes and their corresponding class loaders. Specifically, one or more custom class loaders 104 can be deployed in a virtual machine. Each custom class loader 104 can conform to the Java 1.2 class loader delegation model, and accordingly, can have one or more parent class loaders 106. Additionally, ~~the~~ a virtual machine (not shown) ~~100~~ can have a primordial class loader 108. Importantly, each custom class loader 104, 106 also can be viewed as a peer to each other class loader 104, 106 inasmuch as each class loader 104, 106 can load classes 112, 114 which are dependent upon one another.

Please replace the second paragraph on page 10 which begins with "In accordance with well-known class loading technology" with the following amended paragraph:

In accordance with well-known class loading technology, a class 110 can have a reference to one or more of the class loaders 104. Based upon this reference, the class 110 can request instances of other classes 112, 114. Specifically, the class 110 can request of a custom class loader 104 that the custom class loader 104 instantiate an instance of the requested class 112, 114 based only upon a provided class name. The referenced class loader 104 can forward the request to load the class 112, 114 to specified peer class loaders 104, 106, which in turn can forward the request to their peer parent class loaders 104, 106, and so forth in accordance with the dependency specification of the virtual machine~~100~~.

Please replace the first full paragraph on page 11 which begins with "In order to avoid the inefficiencies" with the following replacement paragraph:

In order to avoid the inefficiencies and potential errors associated with class loading outside of the dependency specification of the virtual machine~~100~~, each custom class loader 104 (as well as the peer class loaders 104, 106, but excepting the primordial class loader 11~~0~~0) can be configured both with a peer class loader list and a dirty bit. Specifically, as illustrated in Figure 2, the class loader 104 can include conventional class loading components, for instance a specific implementation of a findClass() method 202 as well as a reference to a parent class loader 204. Additionally, the class loader 104 can include a peer class loader list 206 and a dirty bit 208. Finally, the class loader 104 can include one or more methods relating to the peer class loader list 206 and dirty bit 208, for instance a generateList() method 210 and an isDirty() method 212.

Please replace the second full paragraph on page 11 which begins with "The peer class loader list 206 can include" with the following replacement paragraph:

The peer class loader list 206 can include a list of peer class loaders which are to be traversed according to the dependency specification of an application within the virtual machine ~~100~~. The dirty bit 208, by comparison, can indicate both when a class has been replaced, in response to which a new class loader must be created and an associated class loader list 206 generated. In one exemplary aspect of the invention, the isDirty() method 212 can expose the status of the dirty bit 208, while the generateList() method 210 can provide logic for creating the class loader list 206.